SOLDAT

How to make a decent climb map

Authors: Viral

Last update: May 21, 2020

Contents

1 Introduction **1**

2 Good practices **1**

3 Polybugs **3**

4 Jumps (climb) **5**

1 Introduction

This document gathers all useful tips, tricks and general mapping rules that the authors of this paper have discovered throughout their beautiful Soldat career. It's supposed to help mappers create high quality maps with minimal effort. Since I'm (Viral) mostly a climb mapper, some sections are strictly related to this gamemode. Despite this fact, it is strongly recommend to read every point of this tutorial. Climb specific aspects has been marked with "(climb)" tag.

This article won't teach you how to use mapping tools or map editors. With this approach the article will hopefully not lose its value over time.

Please note that statements presented here are subjective and based entirely on authors' experience. This means "no way" should be read as "as far as the authors' knowledge goes".

2 Good practices

Keep in mind these are general rules and there is a bunch of reasons not to follow them, some of them will be covered in the following sections. There are a lot of great maps that break some of these rules - the key is to also use your brain when doing it.

Screenshots use common polygon colors: normal - white, deadly - red, ice - blue, green - invisible.

1. **Clean polygon structure**
   Clean structure will greatly lower a chance of polybugs appearing. All of it's components are listed below:

   (a) Use as few polygons, as possible. Creating a square takes 2 polygons, not 10 or 20. What might be even more surprising for beginners - triangle can be created using only one polygon. [1]

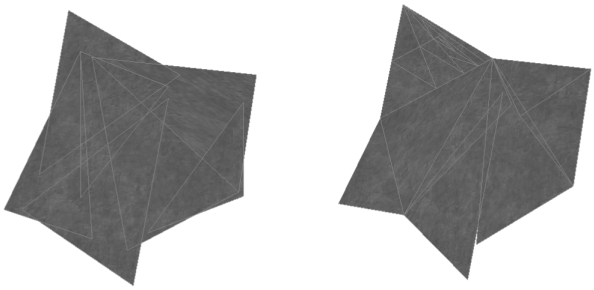

Figure 1: Bad structure on the right, good structure on the left.

If you want to create a complex structure (for better shading possibilities, etc.) use *doesn't collide* polygons, and keep the colliding foundation as simple as possible. [2], [[1]]

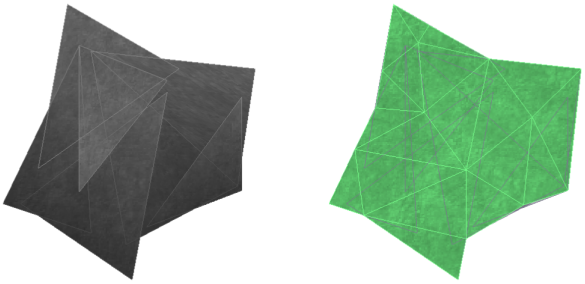

Figure 2: Bad shading on a good structure, good structure before shading on the right. Notice the green *doesn't collide* polygons overlay on the good base structure.

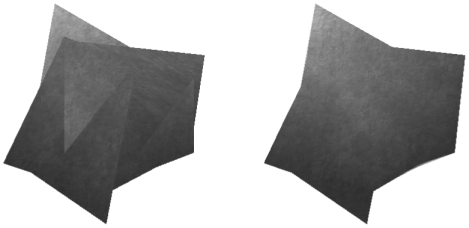

Figure 3: Shading results comparison.

(b) Avoid exposed polygon edges and polygon connections. Fewer of them minimizes the chance of polybugs occurring.

(c) Do not overlap polygons unless your goal is to fix existing polybugs (explained in detail in *Polybugs* section).

(d) Don't use very thin polygons (+/- thinner than soldier's torso). The bigger the polygon, the easier it is to prevent it from affecting a player in an odd, buggy way.

2. **Colors and contrast**
Try to use eye friendly colors, that are easy on the eyes. Using consistent color scheme for different polygon types will make it more bearable for players.

3. **Visuals**
   Those are okay, as long as the sceneries/polygons don't cover up too much. The Inverse Scenery Difficulty Principle goes as follows: The more crucial the part of a map is, the more simpler the visuals, to the point of not having any scenery at all. One well placed scenery will make it more pretty to the eye and won't cripple the gameplay!

4. **Naming convention**
   Use "ctf_" + "(optional) creator's initials/tag_" + "map name". Examples: ctf_mapper_mapname, ctf_randomname. It's also a good idea to keep all the letters lowercase. It's also popular to make the first letter in the map name uppercase: ctf_Examplemap. If u plan on using your own tag, download a mappack beforehand and check if your tag isn't already used by someone with similar nickname.

   Add your full nickname in the map's description, so they won't become anonymous as the time passes.

5. **Consistent difficulty level (climb)**
   Pay attention to the difficulty level of each separate jump or combo. If your goal is to make a noob-friendly map, it might be a good idea to avoid complex, pixel perfect combos. On the other side, you don't want to put 10 repetitive W-spike-jumps in the middle of a speedrun, mechanic-heavy map, just to make it irritating.

6. **Memory/fake maps (climb)**
   Don't create fake platforms that you'll fall through, or platforms that are safe to jump on, interspersed with deadly platforms that look exactly like the safe ones. Once players have memorized the jumps for those, they are really just gimmicky and less than useless.

7. **Shortcuts (climb)**
   Avoid creating shortcuts that cut a lot of time out of otherwise long climb maps. Once discovered (usually in map editors) they become just another really short maps.

   Creating a good shortcut doesn't mean hiding it behind a fake wall, but rather giving a player a high risk/high reward path, where beating a difficult obstacle is rewarded with a shorter cap time.

8. **Spawn points**
   Make sure the player spawn doesn't randomly put players into polygons or kills them. Put it noticeably far from both normal and deadly polygons.

   Flag spawn points should be placed in areas, where they don't move or randomly spawn in different locations (unless there is an idea behind it). Using invisible flag-collide polygons is your friend.

   (climb) If you want to separate blue and red flag spawns, consider how likely it is to find yourself in a situation, where you reached the flag spawn, and someone already took the flag (ruined your run).

   (climb) Don't use more than one alpha/bravo spawn per map, unless it's crucial for the map to be captured. One spawn will allow speedrunners to have consistent times and fair challenge.

3 Polybugs

Knowing how to fix polybugs is just as important as knowing where to look for them! This knowledge also helps when creating jumps based around exploiting them (explained in different section). This section also assumes you maintain clean polygon structure.

It's worth noting, that sometimes applying more than one *possible fix* will give better results, so it's recommended to try mixing them together and seeing what works best.

1. **Corner bounce**
   Occurs when using *non-deadly* polygons in a right, isosceles triangle shape, usually smaller ones. The enemy will most likely hide in the right angle of the poly in question. Ice polygons tend to be less vulnerable, but it's not always the case.

   Possible fixes:

   (a) Use different polygon shape, focus on removing the right angle.

(b) Place deadly polygon near the troubling corner.

(c) Place floor above the polygon, making exploiting of the bug impossible/useless.

(d) Cover the corner with deadly polygon.

(e) Make the poly ice type.

2. **Sticky wall**
   Occurs on all vertical *non-deadly/non-ice* walls. Player can get glued to the wall when falling and hugging it, losing all momentum.

   Possible fixes:

   (a) Tilt the wall slightly, so that it's a bit overhanging. It also lowers or removes the possibility of grabbing the lower corner of the wall (sometimes more or less tricky, but often possible).

   (b) Make the wall *ice*. It might turn into climbable wall instead.

   (c) Make the wall significantly wider. It will most likely turn into climbable wall instead.

3. **Climbable/bouncy wall**
   Occurs on all vertical *non-deadly* walls. Player can fall a bit into the wall and then gets bounced off of it. Good timing allows players to climb/jump on these walls.

   Possible fixes:

   (a) Tilt the wall slightly, so that it's a bit overhanging. It also lowers or removes the possibility of grabbing the lower corner of the wall (sometimes more or less tricky, but often possible).

   (b) Make the wall significantly thinner. It will most likely turn into sticky wall.

4. **Landing on polygons causing bounce**
   Player landing on poly gets stuck in/bounced off the poly. More likely to occur when landing with high momentum.

   Possible fixes:

   (a) Make polygon thicker.

   (b) Move polygons edges as far from the landing zone as possible.

   (c) Transform the polygon in a way that makes player more likely to land on the thicker part of the polygon.

5. **Magnets**
   You know this badboy. Usually ping and netcode will play the biggest part here, but you can try to minimize it.

   Possible fixes:

   (a) Don't connect player's platforms with *deadly* polygons near places where the player is expected to jump.

   (b) Avoid forcing player to move towards *deadly* polygons in very tight places.

6. **Polygons connections causing bounces**
   Occurs when player steps/jumps on the connection of two or more polygons.

   Possible fixes:

   (a) Remove the connection, using the power of your brain.

   (b) Rearrange the polygons, so that the player running from one side to the other starts from the polygon set the furthest to front, ending on the polygon moved the furthest back.

   (c) Cover the edge with (*invisible*) polygon.

   (d) Overlap the polygons, eliminating the connection.

   (e) Add a deadly obstacle just-so-happened to cover the troubling part.

(f) Change the whole structure and give up (seriously, sometimes there's nothing you can do).

Friendly tips: You might wonder how to fix all of these issues and still have a somewhat good looking map. The answer is simple and for once, very easy. **Use *invisible* and *doesn't collide* polygons!**

You want thin polygon, but it bounces? Turn it into a *doesn't collide* polygon and place huge, invisible platform underneath! I don't want to tilt this wall, because i want 90° angles everywhere? Turn it into *doesn't collide* and tilt the *invisible, colliding* wall behind it! No one will notice 10 pixel difference when playing.

4 Jumps (climb)

This section gathers all useful information regarding creation of smooth, uncheesable jumps and more fancy movement setups. Even though creating such jumps is achieved mostly by a lot of in-game testing, it often starts with the same core.

Most importantly, most of the jumps can be either a part of a combo, meaning you require player's momentum from previous jump(s) or static, meaning you expect a player to reset all momentum before the obstacle/don't require any preserved momentum. Therefore core setups presented below will mainly refer to static jumps since taking players momentum into account is map specific and require individual testing.

Speed check mentioned in the text below is a jump following, leading or the discussed jump where a player needs certain preserved momentum to perform it successfully.

This part also assumes you already internalised the content of *Good practices* and *Polybugs* sections.

1. **Cannonball**
   Forcing a player to do a cannonball requires:

   (a) Taking into account, whether the player already has any horizontal momentum, or is capable of getting it from the starting platform.

   If the goal is to make a static cannonball jump, it would be a good idea to also force a player to stop before the jump (resetting his movement speed right before it). Going for a combo cannonball means moving the landing platform further and lower in comparison to the static one.

   (b) The easiest way to prevent player from cheesing static cannonball using *lateflip* is adding a low-hanging ceiling. If you are afraid of cheesing your combo cannonball with long *lateflip* jump, adding a simple speed check on the end of a combo can solve the problem (long late flip is the simplest option). Otherwise moving the landing platform further down and low will eventually make the combo uncheesable even without the need of a speed check.

2. **Reversed (late) cannonball**
   If you want to force a player to do a *reversed (late) cannonball* there is no reliable way of doing that without forcing a static position or using some sort of a guide.

   (a) Notice *lateflip* and *latecannonball* are different in only two points - the moment you perform a flip (*lateflip's* flip is high, losing more momentum, *latecannonball's* flip is flat but gives more momentum). The goal is to either kill player on the flip animation or on the cannonball's speed check.

   (b) You can also use *normal* polygon roof as a guide, to force player into the exact right spot. Pay even more attention to cannonball's speed check.

3. **Semi cannonball**
   Shares the same issues as basic *canonball*. On top of that:

   (a) Placing a *deadly* obstacle between the starting and landing platforms with appropriate height is crucial. Focus mostly on the height near both of the platform, the middle isn't that important and can leave room for *lateflip* cheese.

4. **Booster / 1 pixel jumps**
   It's possible to jump on polygons which have no width or/and height (straight line or point). Jumping on a line is possible on both of it's ends (equivalent to two points).

(a) Since you can't see them in-game, it requires some sort of indicators (sceneries, *doesn't collide* polygons).

(b) Stacking a bunch of these point-polygons in a straight line will give a feeling of a booster. You don't need to make it very dense.

(c) Falling/running parallel to the booster line will make it harder to go through it the more momentum you have.

(d) You can use 1 pixel polygons to fake *walljumps*, make reliable *corner jumps*, *corner grabs* and other stuff! Be creative!

5. **Walljumps**
A wide spectrum of jumps, basically all platforms which a player can jump off of, but staying on them is very hard or close to impossible.

(a) Choosing a wall angle should be dictated by the players' possible ways of approach and their speed. Generally straight vertical wall is a good choice, however using different angles almost always gives more consistent results and fluid movement.

(b) If possible, use *ice* polygons.

(c) Use polygons that are significantly wider than the walljumping side. This rule is even more important when using walls close to vertical or when not using *ice* polygons.

(d) Doing a *wallgrab* (jumping into the direction of the wall when you are on it) can give random results, such as breaking the jump animation, if the wall is vertical or catching the lower edge of the *walljump*. Solve it by adding a stopping wall below and paying attention to player's momentum.

6. **Corner bounce/jumps/grabs**
Even though very similar to *walljumps*, corner variations focus on using only the very corner of the poly, not the entire wall.

(a) Corner grabs/jumps are very precise, but work on every polygon edge. To make it more reliable, you can use *1 pixel jumps* to make it more consistent.

(b) corner bounce works best with a right, isosceles triangle-shaped polygon. Usually quite random and give results similar or worse to bounce polygons. The pictures do show some more reliable setups.

7. **Nade jumps**
Wtf dude.

8. **Combos**
Creating good combo jumps is utilizing every piece of information contained in this guide. It means connecting polybug free, polished jumps together and decorating it with appropriate, gameplay-friendly visuals.

(a) Check each separate jump for mentioned polybugs and make it uncheesable using the knownledge aquired while reading this guide.

(b) Make sure every piece of combo cannot be completed without the preserved momentum acquired from the starting jump.

(c) Place every platform in a spot that allows the combo to be completed flawlessly, without quirky movement if it has been *perfectly* executed. From there add as much error margin as you want, usually by making the platforms wider, but still paying attention to speed checks.

The bigger the error margin, the easier the combo. A fun combo would essentially mean forcing only a chain of jumps, giving broad freedom of movement once in the air. Hard combo would also push a player to pay attention to their movement while mid-air, jumping positions and landing, often going hand in hand with narrow deadly corridors.

(d) It's close to impossible to make a combo (or even a jump) completely polybug free, therefore it's not a bright idea to expect players to execute it perfectly. Keeping this in mind, even the hardest combos should give just enough room for error, to compensate for the game imperfection.

(e) It's always great to test the whole combo from start to finish but sometimes it could be very tedious or nearly impossible. However, most of the combos, especially the longer ones, have stopping/peak points - points where player's momentum resets. This is where you can place testing spawn points, and try individual parts of the combo.